

```

// Called when an ID_CONTROL_NEW_FRAME message is received
// as a result of the frame callback getting a new frame.
void CSnapshotCtrl::OnControlNewframe()
{
    DWORD startTime;           // time (ms since bootup) when we started processing this frame.
    DWORD endTime;
    DWORD msTimeSincePrev;    // time (ms) elapsed since the previous frame was snapped.
    BOOL sendFrame;           // "send this frame as a snapshot"
    BOOL motionOccurred;      // "motion occurred between the previous frame and the current frame."
    BOOL noveltyOccurred;     // "motion occurred between the stable frame and the current frame."
    BOOL changeOccurred;      // "a stable change happened"
    DWORD msIdle;             // time (ms) elapsed since movement was detected.
    int diffThreshold;        // the absolute threshold to use in the difference calculation.
    CRect moveRect;           // the motion rectangle, in video coordinates.
    double pcNumChanged;      // number of pixels that have changed (vs. moved), in percent.
    CTime ctNow;              // the time the frame was taken, in seconds since the epoch.
    CString cStrTag;          // text label for the current picture.
    double fDiff;
    IPLStatus iplStat;
    IplROI *pRoi = NULL;      // region-of-interest for differences of the images (or NULL if none).
    IplROI *prevRoi1, *prevRoi2, *prevRoi3; // previous region-of-interest for images we mess with

    LPCSTR pMsg;              // points to a static error message.
    IplLUT *pLut[3];          // pointers to each lut[] entry.
    IplLUT lut[3];            // the Lookup-table for each channel.
    int lutKey[256 * 3];      // the key values for each channel's LUT.
    int lutValue[256 * 3];    // the value values for each channel's LUT.
    int lutIdx;
    int keyIdx;

    sendFrame = FALSE;
    motionOccurred = FALSE;
    noveltyOccurred = FALSE;
    changeOccurred = FALSE;
    startTime = timeGetTime();
    ctNow = CTime::GetCurrentTime();

    // Keep our callback from overwriting our data.
    m_frameBusy = TRUE;

    // If the video was closed a while ago, ignore this message.
    // If we've already processed this frame, ignore it.
    // This helps when we are capturing frames faster than we can process them.

    if (!m_pFrameCam || !m_pIplIn || !m_readyFrameIn) {
        m_frameBusy = FALSE;
        return;
    }

    // If the camera format is compressed, decompress it first so that IIPL can handle it.
    // Otherwise, just copy it.

    if (m_hicD) {
        if (ICERR_OK != ICDecompress(m_hicD, 0L,
            &m_pFmtCam->bmiHeader, m_pFrameCam,
            &m_pFmtIn->bmiHeader, m_pFrameIn)) {
            Stop();
            AfxMessageBox("Frame decompress failed.");
            m_readyFrameIn = FALSE;
            m_frameBusy = FALSE;
        }
    }
}

```



```

apse
    sendFrame = TRUE;
    break;
case SNAP_ON_MOTION:
case SNAP_ON_NOVELTY:
case SNAP_ON_CHANGE:
    // motion- and change-detection modes are handled below.
    break;
default:    // unknown snap mode.
    Stop();
    ASSERT(FALSE);
}

// If we're going to do any sort of motion/change-detection on this frame,
// do it.

switch (m_snapMode) {
case SNAP_BULB:
case SNAP_TIME_LAPSE:
    // these modes aren't motion modes.
    break;
case SNAP_ON_MOTION:
case SNAP_ON_NOVELTY:
case SNAP_ON_CHANGE:

    // The motion-detection modes require limiting the detection to the motion rectangle.
    // Create a region-of-interest for the motion rectangle.

    pRoi = iplCreateROI(0, moveRect.left, moveRect.top, moveRect.Width(), moveRect.Height());
    if (!pRoi) {
        Stop();
        AfxMessageBox("Couldn't create region-of-interest for motion detection.");
        m_readyFrameIn = FALSE;
        m_frameBusy = FALSE;
        return;
    }

    // Convert the full-color image to an 8-bit/pixel luminance-only image,
    // blur a bit first to lower the noise caused by pixel noise and camera vibration.
    //ZZZ the docs say iplFixedFilter supports in-place, but the routine errored when I tried
to do that.

    iplColorToGray(m_pIplIn, m_pIplTmp);
    iplFixedFilter(m_pIplTmp, m_pIplMotion, IPL_GAUSSIAN_3x3);

    //ZZZ I want to add an edge-enhancement to avoid triggering on shadows or gross lighting c
hanges,
    //ZZZ but don't have time at the moment to build it.

    // If we're just starting,
    // copy this frame to the reference so we don't falsely trigger on motion.

    if (!m_startFrameSeen) {
        iplCopy(m_pIplMotion, m_pIplMotionRef);
    }

    // Find how different the current image is from the motion-detection reference.
    // pixel_is_different = abs(a - b) > (pixel_noise + pixel_noise).
    // This function essentially says "how many corresponding pixels differ by greater than th
e noise in each pixel?"
    //
    // Since iplSubtract() performs saturation arithmetic,

```

```

// we have to perform subtractions and sum their result
// in order to implement our function.

// Limit the region of interest to our motion-detection rectangle,
// and limit the division to its number of pixels.
//ZZZ can we just use iplSetRoi() on the images' ROI's instead of the pointer-setting stuff?

//ZZZ the documentation really isn't explicit about how to set the ROI of an image.

prevRoi1 = m_pIplMotion->roi;
m_pIplMotion->roi = pRoi;
prevRoi2 = m_pIplMotionRef->roi;
m_pIplMotionRef->roi = pRoi;
prevRoi3 = m_pIplTmp->roi;
m_pIplTmp->roi = pRoi;

iplSubtract(m_pIplMotion, m_pIplMotionRef, m_pIplTmp);
iplThreshold(m_pIplTmp, m_pIplTmp, diffThreshold);
fDiff = iplNorm(m_pIplTmp, NULL, IPL_L1) / 255.0;

iplSubtract(m_pIplMotionRef, m_pIplMotion, m_pIplTmp);
iplThreshold(m_pIplTmp, m_pIplTmp, diffThreshold);
fDiff += iplNorm(m_pIplTmp, NULL, IPL_L1) / 255.0;

// Restore the regions of interest.

m_pIplMotion->roi = prevRoi1;
m_pIplMotionRef->roi = prevRoi2;
m_pIplTmp->roi = prevRoi3;

// fDiff is now the number of pixels that differed.
// Convert it to the percent of pixels that differed.

fDiff /= ((double) moveRect.Width() * (double) moveRect.Height());
m_fCurrentMotion = fDiff * 100.0;

// If enough pixels have changed, call it motion.
// Note when our most recent motion occurred
// and that our change-detection algorithm doesn't have to wait for motion to occur any more

if (m_fCurrentMotion > m_fMotionThreshold) {
    motionOccurred = TRUE;
    m_msPrevMoveTime = startTime;
    m_waitingForMovement = FALSE;
}

// Now that we're done with the previous edge-enhanced motion reference,
// update it.

iplCopy(m_pIplMotion, m_pIplMotionRef);

switch (m_snapMode) {
case SNAP_BULB:
case SNAP_TIME_LAPSE:
case SNAP_ON_MOTION:
    // these are not change-relative cases.
    break;

case SNAP_ON_NOVELTY:
case SNAP_ON_CHANGE:

```

姓名	性别	年龄	籍贯	职业	文化程度	健康状况	婚姻状况	子女情况	其他
王德胜	男	45	山东	工人	高中	良好	已婚	2子1女	
李秀英	女	38	河北	教师	大学	良好	已婚	1子1女	
张国强	男	52	江苏	干部	大学	良好	已婚	2子1女	
刘小红	女	28	河南	护士	中专	良好	已婚	1子1女	
陈为民	男	40	浙江	商人	高中	良好	已婚	2子1女	
赵大伟	男	35	湖北	农民	初中	良好	已婚	1子1女	
孙丽娟	女	32	四川	医生	大学	良好	已婚	1子1女	
周建民	男	48	广东	工程师	大学	良好	已婚	2子1女	
吴小芳	女	25	湖南	学生	高中	良好	未婚	无	
郑为民	男	55	安徽	工人	初中	良好	已婚	2子1女	
冯大刚	男	30	江西	干部	大学	良好	已婚	1子1女	
马秀珍	女	42	山西	教师	高中	良好	已婚	2子1女	
徐国强	男	40	福建	商人	高中	良好	已婚	1子1女	
林小红	女	35	广西	护士	中专	良好	已婚	1子1女	
陈为民	男	45	云南	工人	初中	良好	已婚	2子1女	
周建民	男	38	贵州	干部	大学	良好	已婚	1子1女	
吴小芳	女	28	四川	学生	高中	良好	未婚	无	
郑为民	男	50	湖南	工人	初中	良好	已婚	2子1女	
冯大刚	男	32	湖北	干部	大学	良好	已婚	1子1女	
马秀珍	女	40	山西	教师	高中	良好	已婚	2子1女	
徐国强	男	42	福建	商人	高中	良好	已婚	1子1女	
林小红	女	30	广西	护士	中专	良好	已婚	1子1女	
陈为民	男	48	云南	工人	初中	良好	已婚	2子1女	
周建民	男	35	贵州	干部	大学	良好	已婚	1子1女	
吴小芳	女	25	四川	学生	高中	良好	未婚	无	
郑为民	男	52	湖南	工人	初中	良好	已婚	2子1女	
冯大刚	男	30	湖北	干部	大学	良好	已婚	1子1女	
马秀珍	女	40	山西	教师	高中	良好	已婚	2子1女	
徐国强	男	42	福建	商人	高中	良好	已婚	1子1女	
林小红	女	30	广西	护士	中专	良好	已婚	1子1女	
陈为民	男	48	云南	工人	初中	良好	已婚	2子1女	
周建民	男	35	贵州	干部	大学	良好	已婚	1子1女	
吴小芳	女	25	四川	学生	高中	良好	未婚	无	
郑为民	男	52	湖南	工人	初中	良好	已婚	2子1女	
冯大刚	男	30	湖北	干部	大学	良好	已婚	1子1女	
马秀珍	女	40	山西	教师	高中	良好	已婚	2子1女	
徐国强	男	42	福建	商人	高中	良好	已婚	1子1女	
林小红	女	30	广西	护士	中专	良好	已婚	1子1女	
陈为民	男	48	云南	工人	初中	良好	已婚	2子1女	
周建民	男	35	贵州	干部	大学	良好	已婚	1子1女	
吴小芳	女	25	四川	学生	高中	良好	未婚	无	
郑为民	男	52	湖南	工人	初中	良好	已婚	2子1女	
冯大刚	男	30	湖北	干部	大学	良好	已婚	1子1女	
马秀珍	女	40	山西	教师	高中	良好	已婚	2子1女	
徐国强	男	42	福建	商人	高中	良好	已婚	1子1女	
林小红	女	30	广西	护士	中专	良好	已婚	1子1女	
陈为民	男	48	云南	工人	初中	良好	已婚	2子1女	
周建民	男	35	贵州	干部	大学	良好	已婚	1子1女	
吴小芳	女	25	四川	学生	高中	良好	未婚	无	
郑为民	男	52	湖南	工人	初中	良好	已婚	2子1女	

iting for new movement)

ge.

```
if ((pcNumChanged > m_fMotionThreshold) || m_forceSnap) {
```

```

        changeOccurred = TRUE;
        iplCopy(m_pIplMotion, m_pIplChangeRef);
    }

    break;
default:    // unknown snap mode.
    Stop();
    ASSERT(FALSE);
}

// Now we're done with the region of interest for motion-detection.

if (!pRoi) {
    Stop();
    ASSERT(FALSE);
}
iplDeleteROI(pRoi);
pRoi = NULL;

break;
default:    // unknown snap mode.
    Stop();
    ASSERT(FALSE);
}

// Now that we know whether or not there's motion and change,
// decide whether we need to send the image.

switch (m_snapMode) {
case SNAP_BULB:
case SNAP_TIME_LAPSE:
    // These are non-motion modes, handled elsewhere.
    break;
case SNAP_ON_MOTION:
    // If motion happened, send the frame.
    if (motionOccurred) {
        sendFrame = TRUE;
    }
    break;
case SNAP_ON_NOVELTY:
    // If motion occurred relative to the last stable change,
    // send the frame.

    if (noveltyOccurred) {
        sendFrame = TRUE;
    }
    break;
case SNAP_ON_CHANGE:
    // If the image is stable and different enough from the last stable change,
    // send the frame.

    if (changeOccurred) {
        sendFrame = TRUE;
    }
    break;
default:    // unknown snap mode.
    Stop();
    ASSERT(FALSE);
}

// If the user has asked for image histogram equalization, do it.

```

```

// We do this AFTER all motion-detection
// because histogram equalization increases the pixel noise as the image gets more uniform,
// ultimately converting a black image into full-brightness-range noise.

if (m_doHistogramEQ) {
    // initialize the lookup tables for each channel.

    //ASSERT(m_pIplIn->numchannels == 3);
    for (lutIdx = 0; lutIdx < 3; ++lutIdx) {
        pLut[lutIdx] = &lut[lutIdx];

        lut[lutIdx].num = 256;
        lut[lutIdx].key = &lutKey[256 * lutIdx];
        lut[lutIdx].value = &lutValue[256 * lutIdx];
        lut[lutIdx].factor = NULL;
        lut[lutIdx].interpolateType = IPL_LUT_INTER;

        for (keyIdx = 0; keyIdx < 256; ++keyIdx) {
            lutKey[lutIdx * 256 + keyIdx] = keyIdx;
            lutValue[lutIdx * 256 + keyIdx] = 0;
        }
    }

    // Calculate the image histogram,
    // then equalize that histogram and apply it to the image.

    //ZZZ the pre-equalized histogram could be useful for detecting that, for example,
    //ZZZ the image is totally black and isn't worth sending.

    iplComputeHisto(m_pIplIn, &pLut[0]);
    iplHistoEqualize(m_pIplIn, m_pIplIn, &pLut[0]);
}

// If this frame is to be forced, send it.

if (m_forceSnap) {
    sendFrame = TRUE;
}

// If we've decided to send this frame,
// create a fresh IPL copy of it and hand that copy off to the desired window.

if (sendFrame) {
    IplImage *pIpl = NULL;        // the IPL image copy to send away.

    pIpl = iplCloneImage (m_pIplIn);
    if (!pIpl) {
        Stop();
        AfxMessageBox("Couldn't create a duplicate of the captured image");
        m_readyFrameIn = FALSE;
        m_frameBusy = FALSE;
        return;
    }

    // Tag the copy with whatever text the sender wanted.
    // We don't tag the original because the tag would only flicker past in the preview.

    cStrTag = m_labelText;

    if (m_doLabelTime) {
        if (!cStrTag.IsEmpty()) {
            cStrTag += " ";
        }
    }
}

```

```

    }
    CString t = ctNow.Format("%I:%M%p"); // (hr:min am/pm)
    t.MakeLower();
    cStrTag += t;
}

if (m_doLabelDate) {
    if (!cStrTag.IsEmpty()) {
        cStrTag += " ";
    }
    cStrTag += ctNow.Format("%B %d"); // (%B = full month name, %b = abbreviated month name, %d = day of month)
}

// If there's nothing to tag, don't tag the image.

if (!cStrTag.IsEmpty()) {
    pMsg = TagIplImage(pIpl, (LPCSTR) cStrTag, m_labelColorBk, m_labelColorText);
    if (pMsg) {
        Stop();
        AfxMessageBox(pMsg);
        iplDeallocate(pIpl, IPL_IMAGE_HEADER | IPL_IMAGE_DATA);
        pIpl = NULL;
        m_readyFrameIn = FALSE;
        m_frameBusy = FALSE;
        return;
    }
}

// Make this frame "pending".
// If we can't send it right away, we'll send it when we can.
// This arrangement makes motion- and change-detection modes be properly throttled by m_msTimeLapse:
// If motion or change occurred during the time-lapse interval,
// that moved or changed frame is sent once the interval expires.

if (m_pIplPending) {
    iplDeallocate(m_pIplPending, IPL_IMAGE_HEADER | IPL_IMAGE_DATA);
    m_pIplPending = NULL;
}
m_pIplPending = pIpl;
pIpl = NULL; // (since we've moved it to Pending)
}

// If the time-lapse has expired since we last sent a frame (or the user forced this frame),
// and we have a frame to send,
// send or write that pending frame and note the time we sent/wrote it.

if (m_forceSnap) {
    if (!m_pIplPending) { // (forcing a frame had better force the frame into pending state)
        Stop();
        ASSERT(FALSE);
    }
}

if ((m_forceSnap || msTimeSincePrev >= m_msTimeLapse) && m_pIplPending) {
    m_msPrevSnapTime = startTime;

    if (!m_saveToName.IsEmpty()) {
        pMsg = WriteIpl(m_pIplPending, m_saveToName, m_jpegQuality);
        if (pMsg) {
            Stop();
            AfxMessageBox(pMsg);
        }
    }
}

```



```

        iplDeallocate(m_pIplPending, IPL_IMAGE_HEADER | IPL_IMAGE_DATA);
        m_pIplPending = NULL;
        m_readyFrameIn = FALSE;
        m_frameBusy = FALSE;
        return;
    }

    //m_pWndNotify->PostMessage(m_msgNotify, 0, (LPARAM) m_pIplPending); // (a SendMessage() w
could confuse the threads)
    iplDeallocate(m_pIplPending, IPL_IMAGE_HEADER | IPL_IMAGE_DATA);
    m_pIplPending = NULL;

    // Tell our container that an image file is ready for them.
    FireImageReady();
}

// Now we can forget whether the user forced this frame
// and whether this was the first frame after a Start().

m_forceSnap = FALSE;
m_startFrameSeen = TRUE;

// Note how long it took to process this frame.
endTime = timeGetTime();
m_frameTime = endTime - startTime;

// let our preview window know it's time to redraw.
InvalidateControl();

m_readyFrameIn = FALSE; // (because we're done processing the current frame)

m_frameBusy = FALSE;
}

```